

# Package: muttest (via r-universe)

June 13, 2026

**Type** Package

**Title** Mutation Testing

**Version** 0.2.0

**Description** Measure quality of your tests. 'muttest' introduces small changes (mutations) to your code and runs your tests to check if they catch the changes. If they do, your tests are good. If not, your assertions are not specific enough. 'muttest' gives you percent score of how often your tests catch the changes.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Roxygen** list(markdown = TRUE)

**Depends** R (>= 4.1.0)

**Imports** checkmate, cli, fs, mirai, R6, rlang, testthat, treesitter, treesitter.r, withr

**Config/testthat/edition** 3

**URL** <https://jakubsobolewski.com/muttest/>

**Suggests** box, covr, cucumber (>= 2.1.0), ggplot2, knitr, purrr, rmarkdown, shiny, stringr

**VignetteBuilder** knitr

**Config/pak/sysreqs** cmake make libuv1-dev

**Repository** <https://jakubsob.r-universe.dev>

**Date/Publication** 2026-05-14 12:42:20 UTC

**RemoteUrl** <https://github.com/jakubsob/muttest>

**RemoteRef** HEAD

**RemoteSha** b35a34fd32a5480d760421d79e56f5a9e3ee1d4f

## Contents

arithmetic_operators . . . . .	3
boolean_literal . . . . .	4
boolean_literals . . . . .	4
call_name . . . . .	5
comparison_operators . . . . .	6
condition_mutations . . . . .	7
CopyStrategy . . . . .	8
default_copy_strategy . . . . .	8
default_reporter . . . . .	9
default_test_strategy . . . . .	9
delete_statement . . . . .	10
FileTestStrategy . . . . .	10
FullTestStrategy . . . . .	12
index_decrement . . . . .	13
index_increment . . . . .	13
index_mutations . . . . .	14
logical_operators . . . . .	15
MutationReporter . . . . .	16
Mutator . . . . .	19
muttest . . . . .	20
muttest_plan . . . . .	21
na_literal . . . . .	22
na_literals . . . . .	23
negate_condition . . . . .	24
numeric_decrement . . . . .	24
numeric_increment . . . . .	25
numeric_literals . . . . .	25
operator . . . . .	26
PackageCopyStrategy . . . . .	27
ProgressMutationReporter . . . . .	28
remove_condition_negation . . . . .	31
remove_negation . . . . .	32
replace_return_value . . . . .	32
string_empty . . . . .	33
string_fill . . . . .	33
string_literals . . . . .	34
TestStrategy . . . . .	35

---

arithmetic\_operators    *Arithmetic operator mutators*

---

## Description

Returns a ready-made list of `operator()` mutators covering common arithmetic swaps: `+/-`, `*//`, `^/*`, `%%/*`, `%//`.

## Usage

```
arithmetic_operators()
```

## Details

Use on any file that performs calculations. A surviving mutant from this preset typically means an assertion checks a *property* of the result (sign, order) rather than a specific value — replacing `expect_gte()` with `expect_equal()` and a computed expected value usually kills it.

## Value

A list of `operator()` mutators.

## See Also

`vignette("mutators", package = "muttest")` for the full operator table and a worked example showing the direction-insensitive assertion pattern.

`vignette("interpreting-results", package = "muttest")` for how to diagnose survivors and fix the underlying test weakness.

## Examples

```
arithmetic_operators()

## Not run:
plan <- muttest_plan(
  source_files = "R/stats.R",
  mutators = arithmetic_operators()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

---

boolean_literal	<i>Mutate a boolean literal</i>
-----------------	---------------------------------

---

**Description**

Replaces a boolean constant (TRUE, FALSE, T, or F) with another one.

**Usage**

```
boolean_literal(from, to)
```

**Arguments**

from	The literal to be replaced. One of "TRUE", "FALSE", "T", "F".
to	The literal to replace with.

**Examples**

```
boolean_literal("TRUE", "FALSE")
boolean_literal("FALSE", "TRUE")
boolean_literal("T", "F")
boolean_literal("F", "T")
```

---

boolean_literals	<i>Boolean literal mutators</i>
------------------	---------------------------------

---

**Description**

Returns a ready-made list of `boolean_literal()` mutators covering all canonical flips: TRUE/FALSE and T/F.

**Usage**

```
boolean_literals()
```

**Details**

Use on any file that passes or returns boolean flags. A surviving mutant from this preset typically means a test checks a *side effect* of the flag (e.g. the branch taken) rather than the flag value itself — adding `expect_true()/expect_false()` on the return value kills it.

**Value**

A list of `boolean_literal()` mutators.

**See Also**

`vignette("mutators", package = "muttest")` for the full mutator table.

`vignette("interpreting-results", package = "muttest")` for how to diagnose survivors and fix the underlying test weakness.

**Examples**

```
boolean_literals()

## Not run:
plan <- muttest_plan(
  source_files = "R/flags.R",
  mutators = boolean_literals()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

---

call_name	<i>Mutate a function call name</i>
-----------	------------------------------------

---

**Description**

Replaces a function name in a call expression with another name. Useful for swapping semantically related functions such as `any/all`, `min/max`, or `sum/prod`.

**Usage**

```
call_name(from, to)
```

**Arguments**

from	The function name to replace.
to	The function name to replace with.

**Examples**

```
call_name("any", "all")
call_name("min", "max")
call_name("sum", "prod")
```

---

comparison\_operators *Comparison operator mutators*

---

## Description

Returns a ready-made list of `operator()` mutators covering direction swaps (`</>`, `<=/>=`, `==/!=`) and boundary shifts (`</<=`, `>/>=`).

## Usage

```
comparison_operators()
```

## Details

Use on any file with threshold logic, range checks, or filter conditions. A surviving mutant from this preset means the exact boundary value implied by the operator was never passed to the function — adding a test at that boundary value kills it.

## Value

A list of `operator()` mutators.

## See Also

`vignette("mutators", package = "muttest")` for the full operator table and a worked example showing the missing boundary value pattern.

`vignette("interpreting-results", package = "muttest")` for how to diagnose survivors and fix the underlying test weakness.

## Examples

```
comparison_operators()

## Not run:
plan <- muttest_plan(
  source_files = "R/shipping.R",
  mutators = comparison_operators()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

---

condition\_mutations    *Condition mutation mutators*

---

## Description

Returns a ready-made list of `negate_condition()` and `remove_condition_negation()` mutators covering both directions of condition logic: wrapping a plain condition in `!(...)` and stripping `!` from an already-negated condition.

## Usage

```
condition_mutations()
```

## Details

Use on any file with non-trivial `if/while` conditions. Surviving mutants mean the branch outcome was never tested with inputs that cross the boundary — adding a test where the condition flips from `TRUE` to `FALSE` kills them.

## Value

A list of mutators.

## See Also

`vignette("mutators", package = "muttest")` for the full mutator table.

`vignette("interpreting-results", package = "muttest")` for how to diagnose survivors and fix the underlying test weakness.

## Examples

```
condition_mutations()

## Not run:
plan <- muttest_plan(
  source_files = "R/validation.R",
  mutators = condition_mutations()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

---

CopyStrategy

*CopyStrategy interface*

---

### Description

Extend this class to implement a custom copy strategy.

### Methods

#### Public methods:

- [CopyStrategy\\$execute\(\)](#)
- [CopyStrategy\\$clone\(\)](#)

**Method** `execute()`: Copy project files according to the strategy

*Usage:*

`CopyStrategy$execute(original_dir)`

*Arguments:*

`original_dir` The original directory to copy from  
`plan` The current test plan

*Returns:* The path to the temporary directory

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

`CopyStrategy$clone(deep = FALSE)`

*Arguments:*

`deep` Whether to make a deep clone.

### See Also

Other CopyStrategy: [PackageCopyStrategy](#), [default\\_copy\\_strategy\(\)](#)

---

`default_copy_strategy` *Create a default project copy strategy*

---

### Description

Create a default project copy strategy

### Usage

`default_copy_strategy(...)`

**Arguments**

... Arguments passed to the ?PackageCopyStrategy constructor.

**Value**

A ?CopyStrategy object

**See Also**

Other CopyStrategy: [CopyStrategy](#), [PackageCopyStrategy](#)

---

default\_reporter      *Create a default reporter*

---

**Description**

Create a default reporter

**Usage**

default\_reporter(...)

**Arguments**

... Arguments passed to the ?ProgressMutationReporter constructor.

**See Also**

Other MutationReporter: [MutationReporter](#), [ProgressMutationReporter](#)

---

default\_test\_strategy      *Create a default run strategy*

---

**Description**

Create a default run strategy

**Usage**

default\_test\_strategy(...)

**Arguments**

... Arguments passed to the ?FullTestStrategy constructor.

**Value**

A `?TestStrategy` object

**See Also**

Other `TestStrategy`: [FileTestStrategy](#), [FullTestStrategy](#), [TestStrategy](#)

`delete_statement`      *Delete statements one at a time*

**Description**

Produces one mutant per deletable statement, removing each `x <- expr` assignment or standalone `f(...)` call from the source. Surviving mutants reveal untested side effects or dead assignments.

**Usage**

```
delete_statement()
```

**Details**

Function definitions (`x <- function(...){ ... }`) are left untouched to avoid producing structurally broken mutants.

**Value**

A [Mutator](#) object.

**Examples**

```
delete_statement()
```

`FileTestStrategy`      *Run tests matching the mutated source file name*

**Description**

This strategy tells if a mutant is caught by a test matching the source file name.

For example, if the source file name is `foo.R`, and there are test files named `test-foo.R` or `test-bar.R`, only `test-foo.R` will be run.

This strategy should give faster results than `?FullTestStrategy`, especially for big codebases, but the score might be less accurate.

**Super class**

```
mutttest::TestStrategy -> FileTestStrategy
```

## Methods

### Public methods:

- [FileTestStrategy\\$new\(\)](#)
- [FileTestStrategy\\$execute\(\)](#)
- [FileTestStrategy\\$clone\(\)](#)

**Method** `new()`: Initialize the FileTestStrategy

*Usage:*

```
FileTestStrategy$new(  
  load_helpers = TRUE,  
  load_package = c("source", "none", "installed")  
)
```

*Arguments:*

`load_helpers` Whether to load test helpers  
`load_package` The package loading strategy

**Method** `execute()`: Execute the test strategy

*Usage:*

```
FileTestStrategy$execute(path, plan, reporter)
```

*Arguments:*

`path` The path to the test directory  
`plan` The current mutation plan. See `muttest_plan()`.  
`reporter` The reporter to use for test results

*Returns:* The test results

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
FileTestStrategy$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## See Also

Other TestStrategy: [FullTestStrategy](#), [TestStrategy](#), [default\\_test\\_strategy\(\)](#)

---

FullTestStrategy	<i>Run all tests for a mutant</i>
------------------	-----------------------------------

---

### Description

This test strategy tells if a mutant is caught by any test.

To get faster results, especially for big codebases, use `?FileTestStrategy` instead.

### Super class

`muttest::TestStrategy` -> FullTestStrategy

### Methods

#### Public methods:

- `FullTestStrategy$new()`
- `FullTestStrategy$execute()`
- `FullTestStrategy$clone()`

#### Method `new()`: Initialize

*Usage:*

```
FullTestStrategy$new(  
  load_helpers = TRUE,  
  load_package = c("source", "none", "installed")  
)
```

*Arguments:*

`load_helpers` Whether to load test helpers

`load_package` The package loading strategy

#### Method `execute()`: Execute the test strategy

*Usage:*

```
FullTestStrategy$execute(path, plan, reporter)
```

*Arguments:*

`path` The path to the test directory

`plan` The current mutation plan. See `muttest_plan()`.

`reporter` The reporter to use for test results

*Returns:* The test results

#### Method `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
FullTestStrategy$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

**See Also**

Other TestStrategy: [FileTestStrategy](#), [TestStrategy](#), [default\\_test\\_strategy\(\)](#)

---

index_decrement	<i>Decrement subscript indices</i>
-----------------	------------------------------------

---

**Description**

Replaces every simple subscript index in  $x[i]$  or  $x[[i]]$  with  $x[i - 1L] / x[[i - 1L]]$ . Targets identifier and numeric literal indices; complex expressions (e.g.  $x[a + b]$ ) are left untouched.

**Usage**

```
index_decrement()
```

**Value**

A [Mutator](#) object.

**Examples**

```
index_decrement()
```

---

index_increment	<i>Increment subscript indices</i>
-----------------	------------------------------------

---

**Description**

Replaces every simple subscript index in  $x[i]$  or  $x[[i]]$  with  $x[i + 1L] / x[[i + 1L]]$ . Targets identifier and numeric literal indices; complex expressions (e.g.  $x[a + b]$ ) are left untouched.

**Usage**

```
index_increment()
```

**Details**

Catches off-by-one errors where tests never verify the exact element retrieved from a vector or list.

**Value**

A [Mutator](#) object.

**Examples**

```
index_increment()
```

---

index_mutations	<i>Index mutation mutators</i>
-----------------	--------------------------------

---

## Description

Returns a ready-made list of `index_increment()` and `index_decrement()` mutators that shift every simple subscript index by 1.

## Usage

```
index_mutations()
```

## Details

Use on any file that extracts elements from vectors or lists by position. Surviving mutants reveal off-by-one errors where tests only verify that *something* was extracted, not *which* element — asserting the exact value of the extracted element kills them.

## Value

A list of mutators.

## See Also

`vignette("mutators", package = "muttest")` for the full mutator table.

`vignette("interpreting-results", package = "muttest")` for how to diagnose survivors and fix the underlying test weakness.

## Examples

```
index_mutations()

## Not run:
plan <- muttest_plan(
  source_files = "R/selectors.R",
  mutators = index_mutations()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

---

logical\_operators      *Logical operator mutators*

---

## Description

Returns a ready-made list of `operator()` mutators covering short-circuit (`&&/||`) and vectorised (`&/|`) logical operator swaps.

## Usage

```
logical_operators()
```

## Details

Use on any file with compound conditions (`if (a && b)`). A surviving mutant from this preset typically means test inputs are symmetric — both flags TRUE or both FALSE. Adding a test with one flag TRUE and the other FALSE exposes the difference between `&&` and `||` and kills the mutant.

## Value

A list of `operator()` mutators.

## See Also

`vignette("mutators", package = "muttest")` for the full operator table and a worked example showing the symmetric-input pattern.

`vignette("interpreting-results", package = "muttest")` for how to diagnose survivors and fix the underlying test weakness.

## Examples

```
logical_operators()

## Not run:
plan <- muttest_plan(
  source_files = "R/access.R",
  mutators = logical_operators()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

---

MutationReporter      *Reporter for Mutation Testing*

---

## Description

The job of a mutation reporter is to aggregate and display the results of mutation tests. It tracks each mutation attempt, reporting on whether the tests killed the mutation or the mutation survived.

## Public fields

test\_reporter Reporter to use for the testthat::test\_dir function  
 out Output destination for reporter messages  
 width Width of the console in characters  
 unicode Whether Unicode output is supported  
 crayon Whether colored output is supported  
 rstudio Whether running in RStudio  
 hyperlinks Whether terminal hyperlinks are supported  
 current\_file Path of the file currently being mutated  
 current\_mutator Mutator currently being applied  
 plan Complete mutation plan for the test run  
 results List of mutation test results, indexed by file path  
 current\_score Current score of the mutation tests  
 error\_messages List of error messages from failed mutant runs

## Methods

### Public methods:

- [MutationReporter\\$new\(\)](#)
- [MutationReporter\\$start\\_reporter\(\)](#)
- [MutationReporter\\$start\\_file\(\)](#)
- [MutationReporter\\$start\\_mutator\(\)](#)
- [MutationReporter\\$add\\_result\(\)](#)
- [MutationReporter\\$update\(\)](#)
- [MutationReporter\\$end\\_mutator\(\)](#)
- [MutationReporter\\$end\\_file\(\)](#)
- [MutationReporter\\$end\\_reporter\(\)](#)
- [MutationReporter\\$get\\_score\(\)](#)
- [MutationReporter\\$cat\\_tight\(\)](#)
- [MutationReporter\\$cat\\_line\(\)](#)
- [MutationReporter\\$rule\(\)](#)
- [MutationReporter\\$clone\(\)](#)

**Method** new(): Initialize a new reporter

*Usage:*

```
MutationReporter$new(test_reporter = "silent", file = stdout())
```

*Arguments:*

test\_reporter Reporter to use for the testthat::test\_dir function

file Output destination (default: stdout)

**Method** start\_reporter(): Start reporter

*Usage:*

```
MutationReporter$start_reporter(plan = NULL)
```

*Arguments:*

plan The complete mutation plan

temp\_dir Path to the temporary directory for testing

**Method** start\_file(): Start testing a file

*Usage:*

```
MutationReporter$start_file(filename)
```

*Arguments:*

filename Path to the file being mutated

**Method** start\_mutator(): Start testing with a specific mutator

*Usage:*

```
MutationReporter$start_mutator(mutator)
```

*Arguments:*

mutator The mutator being applied

**Method** add\_result(): Add a mutation test result

*Usage:*

```
MutationReporter$add_result(  
  plan,  
  killed,  
  survived,  
  errors,  
  error = NULL,  
  original_code = NULL,  
  mutated_code = NULL  
)
```

*Arguments:*

plan Current testing plan. See muttest\_plan().

killed Whether the mutation was killed by tests

survived Number of survived mutations

errors Number of errors encountered

error Optional error condition from a failed run

original\_code Original source lines before mutation  
mutated\_code Mutated source lines

**Method** update(): Update status (no-op in base class)

*Usage:*

MutationReporter\$update(force = FALSE)

*Arguments:*

force Ignored

**Method** end\_mutator(): End testing with current mutator

*Usage:*

MutationReporter\$end\_mutator()

**Method** end\_file(): End testing current file

*Usage:*

MutationReporter\$end\_file()

**Method** end\_reporter(): End reporter and show summary

*Usage:*

MutationReporter\$end\_reporter()

**Method** get\_score(): Get the current score

*Usage:*

MutationReporter\$get\_score()

**Method** cat\_tight(): Print a message to the output

*Usage:*

MutationReporter\$cat\_tight(...)

*Arguments:*

... Message to print

**Method** cat\_line(): Print a message to the output

*Usage:*

MutationReporter\$cat\_line(...)

*Arguments:*

... Message to print

**Method** rule(): Print a message to the output with a rule

*Usage:*

MutationReporter\$rule(...)

*Arguments:*

... Message to print

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

MutationReporter\$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

**See Also**

Other MutationReporter: [ProgressMutationReporter](#), [default\\_reporter\(\)](#)

---

 Mutator

*Mutator*


---

**Description**

Mutator

Mutator

**Details**

Represents a single code mutation — a pattern to find and a replacement to apply. Every mutator function ([operator\(\)](#), [boolean\\_literal\(\)](#), etc.) returns an instance of this class.

**Public fields**

from The token or operator to replace.

to The replacement token or operator.

query Tree-sitter query used to locate candidate nodes.

match\_fn Optional function(*node\_text*) returning logical; overrides the default *node\_text* == from equality check.

replacement\_fn Optional function(*node\_text*) returning a string; overrides the static *to* value as the replacement text.

mutate\_fn Optional function(*code*) that fully replaces the default mutation logic when set.

**Methods****Public methods:**

- [Mutator\\$new\(\)](#)
- [Mutator\\$mutate\(\)](#)
- [Mutator\\$print\(\)](#)
- [Mutator\\$clone\(\)](#)

**Method** [new\(\)](#): Create a new Mutator.

*Usage:*

```
Mutator$new(
  from,
  to,
  query,
  match_fn = NULL,
  replacement_fn = NULL,
  mutate_fn = NULL
)
```

*Arguments:*

from Token to replace.

to Replacement token.

query Tree-sitter query string.

match\_fn Optional custom match function.

replacement\_fn Optional custom replacement function.

mutate\_fn Optional function(code) that fully overrides the default mutation logic.

**Method** mutate(): Apply this mutator to a character vector of source lines.

*Usage:*

```
Mutator$mutate(code)
```

*Arguments:*

code Character vector of source lines.

*Returns:* A list of mutated code variants (one per match), or NULL if the pattern was not found.

**Method** print(): Print a short summary of the mutator.

*Usage:*

```
Mutator$print()
```

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
Mutator$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

---

```
muttest
```

```
Run a mutation test
```

---

**Description**

Run a mutation test

**Usage**

```
muttest(
  plan,
  path = "tests/testthat",
  reporter = default_reporter(),
  test_strategy = default_test_strategy(),
  copy_strategy = default_copy_strategy(),
  workers = 1,
  timeout = 600 * 1000
)
```

**Arguments**

plan	A mutation testing plan. See <code>muttest_plan()</code> .
path	Path to the test directory.
reporter	Reporter to use for mutation testing results. See <code>?MutationReporter</code> .
test_strategy	Strategy for running tests. See <code>?TestStrategy</code> . The purpose of test strategy is to control how tests are executed. We can run all tests for each mutant, or only tests that are relevant to the mutant.
copy_strategy	Strategy for copying the project. See <code>?CopyStrategy</code> . This strategy controls which files are copied to the temporary directory, where the tests are run.
workers	Number of parallel workers. When greater than 1, mutants are tested concurrently using mirai daemons. Defaults to 1 (sequential).
timeout	Per-mutant timeout in milliseconds. If a mutant's test run exceeds this limit the daemon is interrupted and the result is recorded as an error. Use <code>Inf</code> to disable. Defaults to 600000 (10 minutes).

**Value**

An object of class `muttest_result` containing the overall mutation score.

---

<code>muttest_plan</code>	<i>Create a plan for mutation testing</i>
---------------------------	---

---

**Description**

Each mutant requires rerunning the tests. For large project it might be not feasible to test all mutants in one go. This function allows you to create a plan for selected source files and mutators.

**Usage**

```
muttest_plan(mutators, source_files = fs::dir_ls("R", regexp = "[rR]$"))
```

**Arguments**

mutators	A list of mutators to use. See <code>operator()</code> .
source_files	A vector of file paths to the source files.

**Details**

The plan is in a data frame format, where each row represents a mutant.

You can subset the plan before passing it to the `muttest()` function.

**Value**

A data frame with the test plan. The data frame has the following columns:

- filename: The name of the source file.
- original\_code: The original code of the source file.
- mutated\_code: The mutated code of the source file.
- mutator: The mutator that was applied.

---

na_literal	<i>Mutate an NA or NULL literal</i>
------------	-------------------------------------

---

**Description**

Replaces NA, NULL, or a typed NA constant (NA\_real\_, NA\_integer\_, NA\_complex\_, NA\_character\_) with another value.

**Usage**

```
na_literal(from, to)
```

**Arguments**

from	The literal to replace. One of "NA", "NULL", "NA_real_", "NA_integer_", "NA_complex_", "NA_character_".
to	The replacement literal.

**Details**

In tree-sitter-r, NA and all typed NA variants share the same na node type, while NULL has its own null node type. match\_fn distinguishes between them by comparing the literal text.

**Value**

A [Mutator](#) object.

**Examples**

```
na_literal("NULL", "NA")
na_literal("NA", "NULL")
na_literal("NA", "NA_real_")
na_literal("NA_real_", "NA")
```

---

`na_literals`*NA and NULL literal mutators*

---

## Description

Returns a ready-made list of `na_literal()` mutators covering common swaps between NA, NULL, and the typed NA variants (`NA_real_`, `NA_integer_`, `NA_character_`).

## Usage

```
na_literals()
```

## Details

Use on any file that handles missing values or nullable results. A surviving mutant typically means tests never distinguish NA from NULL, or never check which typed NA is returned — adding `expect_true(is.na(x))` and `expect_equal(class(x), "numeric")` style assertions kills it.

## Value

A list of `na_literal()` mutators.

## See Also

`vignette("mutators", package = "muttest")` for the full mutator table.

`vignette("interpreting-results", package = "muttest")` for how to diagnose survivors and fix the underlying test weakness.

## Examples

```
na_literals()

## Not run:
plan <- muttest_plan(
  source_files = "R/missing.R",
  mutators = na_literals()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

---

negate_condition	<i>Negate the condition of if/while statements</i>
------------------	--

---

**Description**

Wraps the condition expression of each matching statement in `!(...)`. For example, `if (x > 0)` becomes `if (!(x > 0))`.

**Usage**

```
negate_condition(statements = c("if", "while"))
```

**Arguments**

`statements` Character vector of statement types to target. Must be a subset of `c("if", "while")`. Defaults to both.

**Value**

A [Mutator](#) object.

**Examples**

```
negate_condition()
negate_condition(statements = "if")
```

---

numeric_decrement	<i>Decrement numeric literals</i>
-------------------	-----------------------------------

---

**Description**

Replaces every numeric literal `n` with `n - by`. Handles both integer (e.g. `5L`) and floating-point (e.g. `3.14`) literals.

**Usage**

```
numeric_decrement(by = 1)
```

**Arguments**

`by` The amount to subtract. Defaults to 1.

**Value**

A [Mutator](#) object.

**Examples**

```
numeric_decrement()
numeric_decrement(by = 2)
```

---

numeric\_increment      *Increment numeric literals*

---

**Description**

Replaces every numeric literal *n* with *n + by*. Handles both integer (e.g. 5L) and floating-point (e.g. 3.14) literals.

**Usage**

```
numeric_increment(by = 1)
```

**Arguments**

*by*                      The amount to add. Defaults to 1.

**Value**

A [Mutator](#) object.

**Examples**

```
numeric_increment()
numeric_increment(by = 2)
```

---

numeric\_literals      *Numeric literal mutators*

---

**Description**

Returns a ready-made list of [numeric\\_increment\(\)](#) and [numeric\\_decrement\(\)](#) mutators that shift every numeric literal by 1.

**Usage**

```
numeric_literals()
```

**Details**

Use on any file with numeric constants used as thresholds, counts, or offsets. A surviving mutant means tests never verify the exact value of the constant — asserting the precise numeric result rather than a property (e.g. `sign`) kills it.

**Value**

A list of mutators.

**See Also**

`vignette("mutators", package = "muttest")` for the full mutator table.

`vignette("interpreting-results", package = "muttest")` for how to diagnose survivors and fix the underlying test weakness.

**Examples**

```
numeric_literals()

## Not run:
plan <- muttest_plan(
  source_files = "R/thresholds.R",
  mutators = numeric_literals()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

---

operator

*Mutate a binary operator*

---

**Description**

Produces one mutant per occurrence of `from` in the source file, replacing it with `to`. A surviving mutant means your tests cannot distinguish the original operator from the replacement — pointing at the missing assertion or input value.

**Usage**

```
operator(from, to)
```

**Arguments**

<code>from</code>	The operator to replace (e.g. <code>"+"</code> , <code>"=="</code> , <code>"&gt;"</code> ).
<code>to</code>	The replacement operator.

**Details**

Use this when you need a specific swap not covered by the preset collections ([arithmetic\\_operators\(\)](#), [comparison\\_operators\(\)](#), [logical\\_operators\(\)](#)).

**Value**

A [Mutator](#) object.

**See Also**

[comparison\\_operators\(\)](#), [arithmetic\\_operators\(\)](#), [logical\\_operators\(\)](#) for ready-made preset lists.

[vignette\("mutators", package = "muttest"\)](#) for the full operator reference with examples of what each preset catches.

[vignette\("interpreting-results", package = "muttest"\)](#) to learn how to read surviving mutants and strengthen the tests they expose.

**Examples**

```
operator("+", "-")
operator("=", "!=")
operator(">", ">=") # probe the strict vs. non-strict boundary
```

---

PackageCopyStrategy    *Package copy strategy*

---

**Description**

It copies all files and directories from the original directory to a temporary directory.

**Super class**

[muttest::CopyStrategy](#) -> PackageCopyStrategy

**Methods****Public methods:**

- [PackageCopyStrategy\\$execute\(\)](#)
- [PackageCopyStrategy\\$clone\(\)](#)

**Method** `execute()`: Copy project files, excluding hidden and temp directories

*Usage:*

```
PackageCopyStrategy$execute(original_dir, plan)
```

*Arguments:*

`original_dir` The original directory to copy from  
`plan` The current test plan

*Returns:* The path to the temporary directory

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
PackageCopyStrategy$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

**See Also**

Other CopyStrategy: [CopyStrategy](#), [default\\_copy\\_strategy\(\)](#)

---

ProgressMutationReporter

*Progress Reporter for Mutation Testing*

---

**Description**

A reporter that displays a progress indicator for mutation tests. It provides real-time feedback on which mutants are being tested and whether they were killed by tests.

**Super class**

[muttest::MutationReporter](#) -> ProgressMutationReporter

**Public fields**

`start_time` Time when testing started (for duration calculation)

`min_time` Minimum test duration to display timing information

`col_config` List of column configuration for report formatting

`survived_detail` Controls how survived mutants are reported (summary, none)

`survived_mutants` List to store details of survived mutants for summary reporting

**Methods****Public methods:**

- [ProgressMutationReporter\\$format\\_column\(\)](#)
- [ProgressMutationReporter\\$fmt\\_h\(\)](#)
- [ProgressMutationReporter\\$fmt\\_r\(\)](#)
- [ProgressMutationReporter\\$new\(\)](#)
- [ProgressMutationReporter\\$start\\_reporter\(\)](#)
- [ProgressMutationReporter\\$add\\_result\(\)](#)
- [ProgressMutationReporter\\$update\(\)](#)
- [ProgressMutationReporter\\$end\\_file\(\)](#)
- [ProgressMutationReporter\\$cr\(\)](#)
- [ProgressMutationReporter\\$end\\_reporter\(\)](#)
- [ProgressMutationReporter\\$print\(\)](#)
- [ProgressMutationReporter\\$clone\(\)](#)

**Method** `format_column()`: Format a column with specified padding and width

*Usage:*

`ProgressMutationReporter$format_column(text, col_name, colorize = NULL)`

*Arguments:*

text Text to format  
 col\_name Column name to use configuration from  
 colorize Optional function to color the text

**Method** `fmt_h()`: Format the header of the report

*Usage:*

`ProgressMutationReporter$fmt_h()`

**Method** `fmt_r()`: Format a row of the report

*Usage:*

`ProgressMutationReporter$fmt_r(status, k, s, e, t, score, mutator, file)`

*Arguments:*

status Status symbol (e.g., tick or cross)  
 k Number of killed mutations  
 s Number of survived mutations  
 e Number of errors  
 t Total number of mutations  
 score Score percentage  
 mutator The mutator used  
 file The file being tested

*Returns:* Formatted row string

**Method** `new()`: Initialize a new progress reporter

*Usage:*

```
ProgressMutationReporter$new(
  test_reporter = "silent",
  min_time = 1,
  file = stdout(),
  survived_detail = c("summary", "none")
)
```

*Arguments:*

test\_reporter Reporter to use for `testthat::test_dir`  
 min\_time Minimum time to show elapsed time (default: 1s)  
 file Output destination (default: stdout)  
 survived\_detail Controls how survived mutants are reported. One of "summary" (default) or "none".

**Method** `start_reporter()`: Start reporter

*Usage:*

`ProgressMutationReporter$start_reporter(plan = NULL)`

*Arguments:*

plan The complete mutation plan

**Method** `add_result()`: Add a mutation test result

*Usage:*

```
ProgressMutationReporter$add_result(
  plan,
  killed,
  survived,
  errors,
  error = NULL,
  original_code = NULL,
  mutated_code = NULL
)
```

*Arguments:*

`plan` Current testing plan. See `muttest_plan()`.  
`killed` Whether the mutation was killed by tests  
`survived` Number of survived mutations  
`errors` Number of errors encountered  
`error` Optional error condition from a failed run  
`original_code` Original source lines before mutation  
`mutated_code` Mutated source lines

**Method** `update()`: Update status spinner (for long-running operations)

*Usage:*

```
ProgressMutationReporter$update(force = FALSE)
```

*Arguments:*

`force` Force update even if interval hasn't elapsed

**Method** `end_file()`: End testing current file

*Usage:*

```
ProgressMutationReporter$end_file()
```

**Method** `cr()`: Carriage return if dynamic, newline otherwise

*Usage:*

```
ProgressMutationReporter$cr()
```

**Method** `end_reporter()`: End reporter with detailed summary

*Usage:*

```
ProgressMutationReporter$end_reporter()
```

**Method** `print()`: Print the mutation test result with survived diffs

*Usage:*

```
ProgressMutationReporter$print()
```

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
ProgressMutationReporter$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

**See Also**

Other MutationReporter: [MutationReporter](#), [default\\_reporter\(\)](#)

---

remove\_condition\_negation

*Remove negation from the condition of if/while statements*

---

**Description**

The inverse of [negate\\_condition\(\)](#). Strips the leading ! from any already-negated condition, so `if (!done)` becomes `if (done)` and `while (!ready)` becomes `while (ready)`.

**Usage**

```
remove_condition_negation(statements = c("if", "while"))
```

**Arguments**

`statements` Character vector of statement types to target. Must be a subset of `c("if", "while")`. Defaults to both.

**Details**

Unlike [remove\\_negation\(\)](#), this mutator is scoped exclusively to conditions, leaving negations in other positions (assignments, return values, etc.) untouched.

**Value**

A [Mutator](#) object.

**Examples**

```
remove_condition_negation()
remove_condition_negation(statements = "while")
```

---

remove_negation	<i>Remove logical negation</i>
-----------------	--------------------------------

---

**Description**

Removes the ! unary operator from an expression. For example, !is.na(x) becomes is.na(x) and !(a > b) becomes (a > b).

**Usage**

```
remove_negation()
```

**Value**

A [Mutator](#) object.

**Examples**

```
remove_negation()
```

---

replace_return_value	<i>Replace the value in explicit return() calls</i>
----------------------	---

---

**Description**

Replaces the argument of every return(expr) with a fixed value (default "NULL"). Tests that only check that a function returns *something* without asserting the value will not kill these mutants.

**Usage**

```
replace_return_value(replacement = "NULL")
```

**Arguments**

replacement	Raw R source text to substitute as the return value. Defaults to "NULL". Examples: "NA" inserts the missing value NA; "'NULL'" (inner quotes) inserts the string "NULL"; "'NA'" inserts the string "NA" rather than the missing value.
-------------	--

**Details**

Only explicit return() calls are targeted. Implicit returns (the last expression of a function body) are not affected.

**Value**

A [Mutator](#) object.

**Examples**

```
replace_return_value()
replace_return_value("NA")
```

---

string_empty	<i>Mutate non-empty string literals to the empty string</i>
--------------	---

---

**Description**

Replaces any non-empty string literal in the source code with "". The empty string itself is not mutated (use [string\\_fill\(\)](#) for that).

**Usage**

```
string_empty()
```

**Value**

A [Mutator](#) object.

**Examples**

```
string_empty()
```

---

string_fill	<i>Mutate the empty string literal to a placeholder string</i>
-------------	--

---

**Description**

Replaces "" with a fill string (default "mutant") so that code paths that depend on an empty string can be detected.

**Usage**

```
string_fill(fill = "mutant")
```

**Arguments**

fill	The replacement string. Defaults to "mutant". Override when the codebase already contains "mutant" as a meaningful value.
------	---

**Value**

A [Mutator](#) object.

**Examples**

```
string_fill()
string_fill(fill = "PLACEHOLDER")
```

---

string_literals	<i>String literal mutators</i>
-----------------	--------------------------------

---

### Description

Returns a ready-made list of `string_empty()` and `string_fill()` mutators covering both directions: collapsing non-empty strings to "" and filling empty strings with a placeholder.

### Usage

```
string_literals()
```

### Details

Use on any file where string values are passed to downstream logic or returned to callers. Surviving mutants reveal tests that only check *type* or *length* — asserting the exact string content kills them.

### Value

A list of mutators.

### See Also

`vignette("mutators", package = "muttest")` for the full mutator table.

`vignette("interpreting-results", package = "muttest")` for how to diagnose survivors and fix the underlying test weakness.

### Examples

```
string_literals()

## Not run:
plan <- muttest_plan(
  source_files = "R/labels.R",
  mutators = string_literals()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

---

TestStrategy	<i>TestStrategy interface</i>
--------------	-------------------------------

---

## Description

Extend this class to implement a custom test strategy.

## Methods

### Public methods:

- [TestStrategy\\$execute\(\)](#)
- [TestStrategy\\$clone\(\)](#)

**Method** `execute()`: Execute the test strategy

*Usage:*

```
TestStrategy$execute(path, plan, reporter)
```

*Arguments:*

`path` The path to the test directory

`plan` The current mutation plan. See `muttest_plan()`.

`reporter` The reporter to use for test results

*Returns:* The test result

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
TestStrategy$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## See Also

Other TestStrategy: [FileTestStrategy](#), [FullTestStrategy](#), [default\\_test\\_strategy\(\)](#)

# Index

- \* **CopyStrategy**
  - CopyStrategy, 8
  - default\_copy\_strategy, 8
  - PackageCopyStrategy, 27
- \* **MutationReporter**
  - default\_reporter, 9
  - MutationReporter, 16
  - ProgressMutationReporter, 28
- \* **TestStrategy**
  - default\_test\_strategy, 9
  - FileTestStrategy, 10
  - FullTestStrategy, 12
  - TestStrategy, 35
- arithmetic\_operators, 3
- arithmetic\_operators(), 26, 27
- boolean\_literal, 4
- boolean\_literal(), 4, 19
- boolean\_literals, 4
- call\_name, 5
- comparison\_operators, 6
- comparison\_operators(), 26, 27
- condition\_mutations, 7
- CopyStrategy, 8, 9, 28
- default\_copy\_strategy, 8, 8, 28
- default\_reporter, 9, 19, 31
- default\_test\_strategy, 9, 11, 13, 35
- delete\_statement, 10
- FileTestStrategy, 10, 10, 13, 35
- FullTestStrategy, 10, 11, 12, 35
- index\_decrement, 13
- index\_decrement(), 14
- index\_increment, 13
- index\_increment(), 14
- index\_mutations, 14
- logical\_operators, 15
- logical\_operators(), 26, 27
- MutationReporter, 9, 16, 31
- Mutator, 10, 13, 19, 22, 24–26, 31–33
- muttest, 20
- muttest::CopyStrategy, 27
- muttest::MutationReporter, 28
- muttest::TestStrategy, 10, 12
- muttest\_plan, 21
- na\_literal, 22
- na\_literal(), 23
- na\_literals, 23
- negate\_condition, 24
- negate\_condition(), 7, 31
- numeric\_decrement, 24
- numeric\_decrement(), 25
- numeric\_increment, 25
- numeric\_increment(), 25
- numeric\_literals, 25
- operator, 26
- operator(), 3, 6, 15, 19, 21
- PackageCopyStrategy, 8, 9, 27
- ProgressMutationReporter, 9, 19, 28
- remove\_condition\_negation, 31
- remove\_condition\_negation(), 7
- remove\_negation, 32
- remove\_negation(), 31
- replace\_return\_value, 32
- string\_empty, 33
- string\_empty(), 34
- string\_fill, 33
- string\_fill(), 33, 34
- string\_literals, 34
- TestStrategy, 10, 11, 13, 35